UNIVERSITATEA “LUCIAN BLAGA” DIN SIBIU

FACULTATEA DE INGINERIE

DEPARTAMENTUL DE CALCULATOARE ŞI INGINERIE ELECTRICĂ

**Proiect la disciplina Procesarea Numerică a Semnalelor**

**Procesarea unui semnal audio vocal**

Masterand: Ignat Codrina-Victoria

Aplicații Avansate în Ingineria Electrică, anul I

Cuprins

[1. Structura proiectului – explicații generale 3](#_Toc92314532)

[2. Funcția de downsampling a semnalului 4](#_Toc92314533)

[3. Funcția de upsampling a semnalului 4](#_Toc92314534)

[4. Funcțiile de filtrare 5](#_Toc92314535)

[5. Versiunile proiectului și rezultatele evaluărilor 7](#_Toc92314536)

[5.1. Algoritmii utilizați 7](#_Toc92314537)

[5.2. Versiunea V01 – alegerea filtrului FTJ 8](#_Toc92314538)

[5.3. Versiunea V02 14](#_Toc92314539)

[5.4. Versiunea V03 – explicații 14](#_Toc92314540)

[5.5. Versiunea V04 – explicații 16](#_Toc92314541)

[5.6. Versiunea V05 – explicații + alegerea filtrului FTB 17](#_Toc92314542)

[6. Concluzii 19](#_Toc92314543)

[7. Anexe 20](#_Toc92314544)

[8. Bibliografie 29](#_Toc92314545)

# Structura proiectului – explicații generale

Scopul proiectului este procesarea unui semnal audio prin diverse tehnici, downsampling, upsampling, filtrare.

Din cauza faptului că algoritmii aleși de verificare a calității semnalelor audio cer aceeași frecvență de eșantionare pentru ambele semnale comparate, proiectul realizat are până acum 5 versiuni. Primele două versiuni prezintă diferențe minore de cod (prima are filtrul implementat în fișierul de bază al proiectului, a doua are filtrarea implementată ca funcție separată și apelată doar în fișierul de bază), iar V03, V04 și V05 au fost ajustate astfel încât să se poată realiza comparația dintre semnalele audio din diverse etape ale proiectului folosind algoritmii aleși. Detalierea fiecărei versiuni se găsește în capitolul 5 al prezentei documentații.

Numele variabilelor de interes din proiect:

* y = variabila în care se salvează semnalul audio original
* Fs = variabila în care se salvează frecvența de eșantionare a semnalului original
* factor\_down = factor de downsampling
* factor\_up = factor de upsampling
* Fs\_down = frecvența de eșantionare a semnalului căruia i s-a aplicat downsampling; Fs\_down = Fs/factor\_down;
* Fs\_up = frecvența de eșantionare a semnalului căruia i s-a aplicat upsampling; Fs\_up = Fs\_down \* factor\_up;
* y2 = variabila în care se salvează semnalul audio downsampled; y2 = myDownsamplingFunction(y\_filtrat,Fs\_down,factor\_down);
* y3 = variabila în care se salvează semnalul audio upsampled; y3 = myUpsamplingFunction(y2,Fs\_up,factor\_up);
* y3\_filtered = semnalul upsampled, filtrat.

Numele funcțiilor din proiect:

* myDownsamplingFunction = functia de downsampling;
* myUpsamplingFunction = functia de upsampling;
* LowPassFilter = funcția pentru filtrul trece-jos;
* BandPassFilter = funcția pentru filtrul trece-bandă;
* Proiect\_PNS = fișierul de bază al proiectului.

# Funcția de downsampling a semnalului

Cod:

**myDownsamplingFunction.m**

function y2 = myDownsamplingFunction(y\_filtrat, Fs\_down, factor\_down)

y2 = y\_filtrat(1:factor\_down:end);

audiowrite('Downsampled\_signal.wav',y2,Fs\_down);

end

Funcția pune în y2 fiecare al **factor\_down**-lea element al **y\_filtrat** (semnalul original, filtrat înainte de downsampling), de la primul element și până la finalul vectorului, apoi scrie în fișierul „Downsampled\_signal.wav” sunetul rezultat în urma decimării. Acest sunet va fi folosit în comparația cu semnalul de Fs = 16 kHz, folosind algoritmii aleși.

# Funcția de upsampling a semnalului

Cod:

**myUpsamplingFunction.m**

function y3 = myUpsamplingFunction(y2,Fs\_up,factor\_up)

y3 = zeros(1,2\*length(y2));

for i = 1:length(y2)

for j = 1:factor\_up

y3(i\*factor\_up - j + 1)=y2(i);

end

end

end

Vectorul de zero-uri y3 a fost creat pentru a aloca memorie viitorului semnal ce urmează a fi creat prin upsampling. Cele două bucle care urmează parcurg semnalul inițial și repetă fiecare eșantion de **factor\_up** ori în **y3**, care reprezintă semnalul după upsampling.

Prima versiune a funcției a fost cea de mai jos, care nu putea primi însă ca parametru orice factor întreg de upsampling:

for i = 1:length(y2)

y3(i\*2)=y2(i);

y3(i\*2-1)=y2(i);

end

Astfel, se realiza upsampling doar pentru factorul 2.

# Funcțiile de filtrare

Pentru a vedea dacă se modifică rezultatele verificării calității sunetului, am folosit două filtre: Trece-Jos și Trece-Bandă.

Prima dată am folosit **filtrul trece jos** (primele 4 versiuni ale proiectului) atât la filtrarea semnalului înainte de downsampling, dar și după upsampling, cu frecvența de tăiere 4000 Hz în ambele cazuri, deoarece:

* Având în vedere frecvența de eșantionare de 8000 Hz a noului semnal rezultat în urma downsampling, componentele de peste 4000 Hz nu ar trebui să mai existe în spectru, pentru a respecta teorema eșantionării, și pentru a nu altera sunetul rezultat cu componente nedorite.
* Spectrul vocii umane este suficient acoperit până în 4000 Hz.
* După upsampling, având în vedere că semnalul construit provine dintr-un semnal cu frecvența de eșantionare de 8000 Hz, am ales tot frecvența de tăiere 4000 Hz. Fără filtru, peste sunetul vocii se suprapuneau componente de frecvență înaltă, care se vedeau în spectru, dar se și auzeau sub forma unui sunet ascuțit.
* Filtrul a fost ales pentru a înlătura efectul de aliere.

Codul aferent implementării filtrului trece jos :

function LPF = LowPassFilter (Ft, y, Fs)

N = 5000;

h = zeros(1,N);

for n=-(N-1)/2 : (N-1)/2

h(n+(N-1)/2 +1)=h(n+(N-1)/2+1)+(2\*Ft/Fs)\*sinc(2\*n\*Ft/Fs);

end

LPF = conv(y,h);

end

N reprezintă numărul de coeficienți ai filtrului. În bucla for sunt calculați coeficienții filtrului. Funcția conv calculează convoluția dintre semnalul y și coeficienții h, ceea ce înseamnă filtrarea propriu-zisă a semnalului y.

În versiunea V05 a proiectului, am folosit un filtru trece-bandă, atât înainte de downsampling, cât și după upsampling, tot pentru motivele enumerate mai sus, dar și pentru a vedea dacă există vreo diferență la rezultatele algoritmilor de calitate a sunetului. De asemenea, am considerat că astfel elimin unele componente de frecvență joasă care nu ar trebui să apară în spectru.

Cod:

function BPF = BandPassFilter (Ft1, Ft2 , y, Fs)

N = 5000;

h = zeros(1,N);

for n=-(N-1)/2 : (N-1)/2

h(n+(N-1)/2 +1)=h(n+(N-1)/2+1)+(2\*Ft2/Fs)\*sinc(2\*n\*Ft2/Fs)-(2\*Ft1/Fs)\*sinc(2\*n\*Ft1/Fs);

end

BPF = conv(y,h);

End

N reprezintă numărul de coeficienți ai filtrului. În bucla for sunt calculați coeficienții filtrului, dar după altă formulă față de cazul anterior, deoarece și tipul de filtru diferă. Funcția conv calculează convoluția dintre semnalul y și coeficienții h, ceea ce înseamnă filtrarea propriu-zisă a semnalului y.

# Versiunile proiectului și rezultatele evaluărilor

## 5.1. Algoritmii utilizați

## 

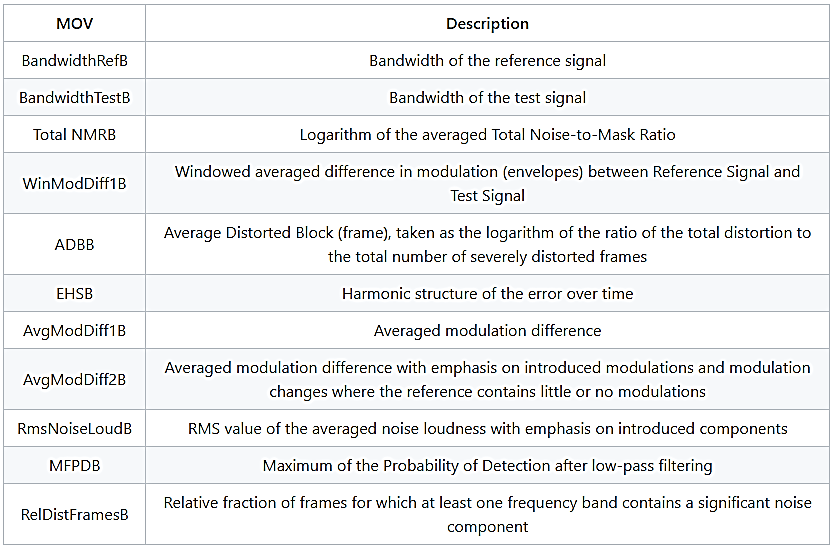
Algoritmii utilizați pentru evaluarea perceptuală a calității sunetului sunt: PEAQ și ViSQOL, despre care sunt prezentate câteva lucruri în continuare.

**Perceptual Evaluation of Audio Quality (PEAQ)** este un algoritm standardizat pentru măsurarea perceptuală obiectivă a calității sunetului, dezvoltat între anii 199401998 de un grup de experți din cadrul ITU-R (International Telecommunication Union's Radiocommunication Sector. Utilizează software care simulează proprietățile perceptuale ale urechii umane. PEAQ caracterizează calitatea audio ca și când ar fi percepută de către subiecții unui test de ascultare. Rezultatele PEAQ reprezintă scoruri între 1 (rău) și 5 (excelent).

În cazul algoritmului PEAQ utilizat pentru verificarea proiectului, rezultatul este dat de valoarea ODG (Objective Difference Grade), o valoare cuprinsă între 0 și -4, corespondența dintre rezultatele ITU-R și ODG fiind:

|  |  |  |
| --- | --- | --- |
| Impairment description | ITU-R Grade | ODG |
| Imperceptible | 5.0 | 0.0 |
| Perceptible, but not annoying | 4.0 | -1.0 |
| Slightly annoying | 3.0 | -2.0 |
| Annoying | 2.0 | -3.0 |
| Very annoying | 1.0 | -4.0 |

De asemenea, algoritmul oferă și modelul variabilelor de ieșire (MOVs – Model Output Variables), variabile care sunt combinate pentru a obține scorul final al PEAQ. Cei 11 parametri oferiți de rezultatul MOVs sunt:



**ViSQOL (Virtual Speech Quality Objective Listener)** este un model de evaluare obiectivă a calității vocii, model bazat pe semnal, care modelează perceptual calitatea discursului uman folosind măsura spectro-temporală a similarității dintre un semnal de referință și un semnal de test. Această metrică a fost creată în special pentru a fi robustă la problemele de calitate asociate cu transmisia VoIP (Voice over IP). ViSQOL este un algoritm similar cu POLQA. ViSQOL se focusează pe similaritățile dintre referință și semnalul de test folosind o metrică numită NSIM – Neurogram Similarity Index Measure. Această metrică este tradusă într-un scor cuprins între 1 (cel mai rău caz) și 5 (cel mai bun caz), numit MOS-LQO (Mean Opinion Score – Listening Quality Objective).

## 5.2. Versiunea V01 – alegerea filtrului FTJ

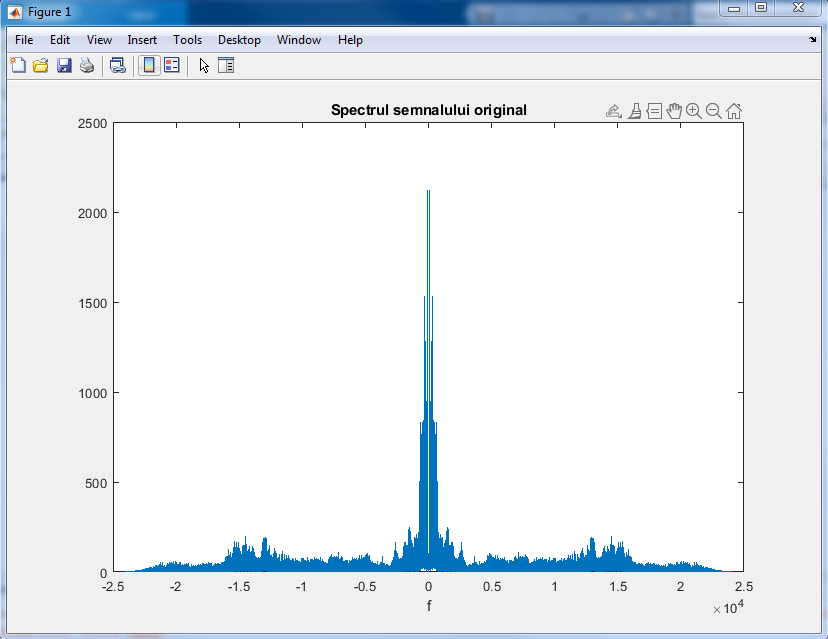
Versiunea aceasta a proiectului reprezintă o versiune simplă, cu filtrul FTJ implementat înainte de downsampling dar și după upsampling, cu implementarea filtrului în corpul fișierului principal.

Opțiunea alegerii filtrului tip trece-jos este justificată în capitolul 4.

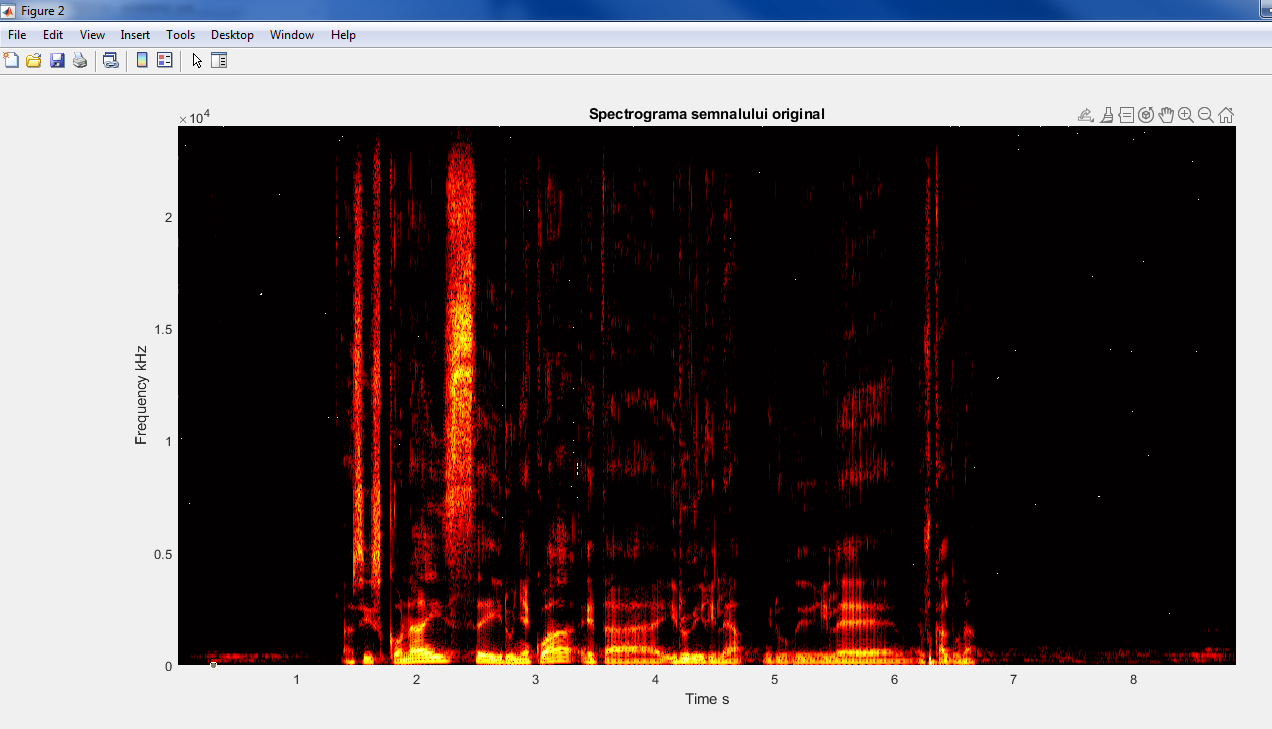
Codul pentru versiunea V01 a proiectului se găsește în Anexa 1.

Etapele pe care le-am parcurs în realizarea proiectului:

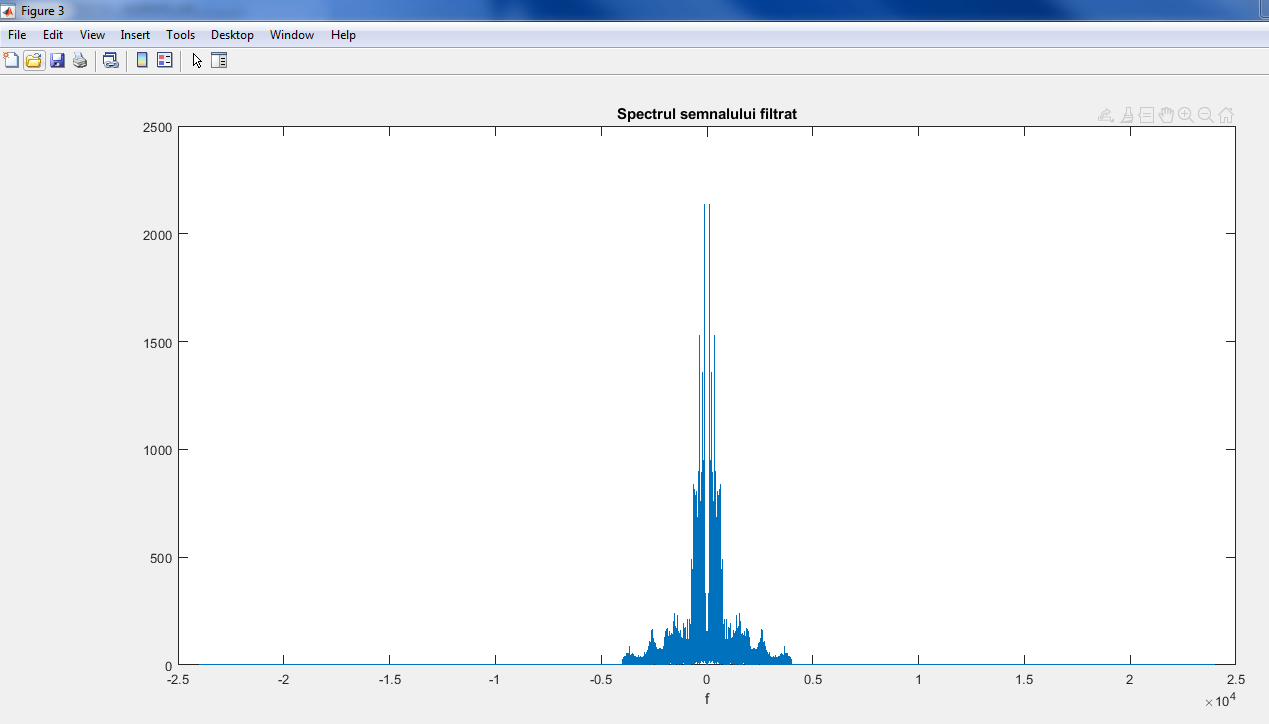
1. Încărcarea semnalului audio original cu funcția audioread.
2. Declararea variabilelor, respectiv a factorilor de downsampling și upsampling.
3. Calculul frecvențelor de eșantionare a semnalelor după downsampling și upsampling, respectiv după formulele: Fs\_down = Fs/factor\_down și Fs\_up = Fs/factor\_up.
4. Afișarea spectrului semnalului original:



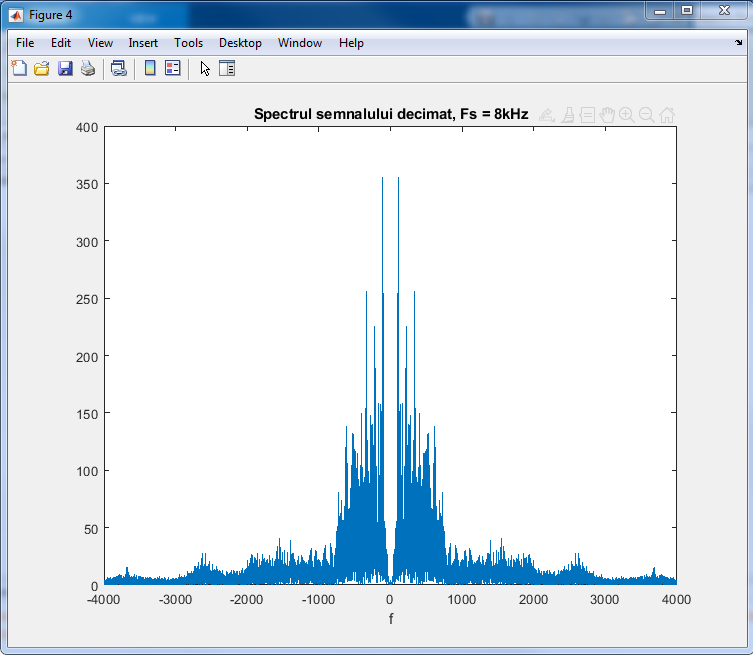
1. Afișarea spectrogramei semnalului original:



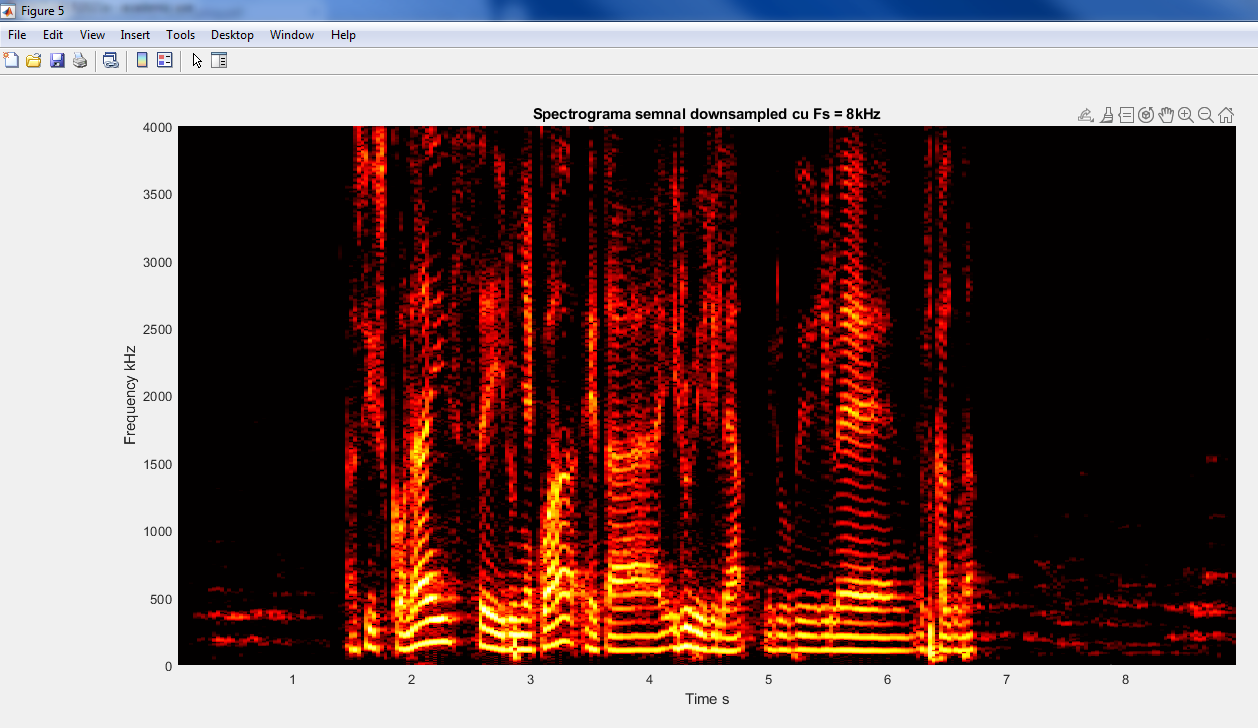
1. Implementarea filtrului : calculul coeficienților și convoluția.
2. Afișarea spectrului semnalului filtrat:



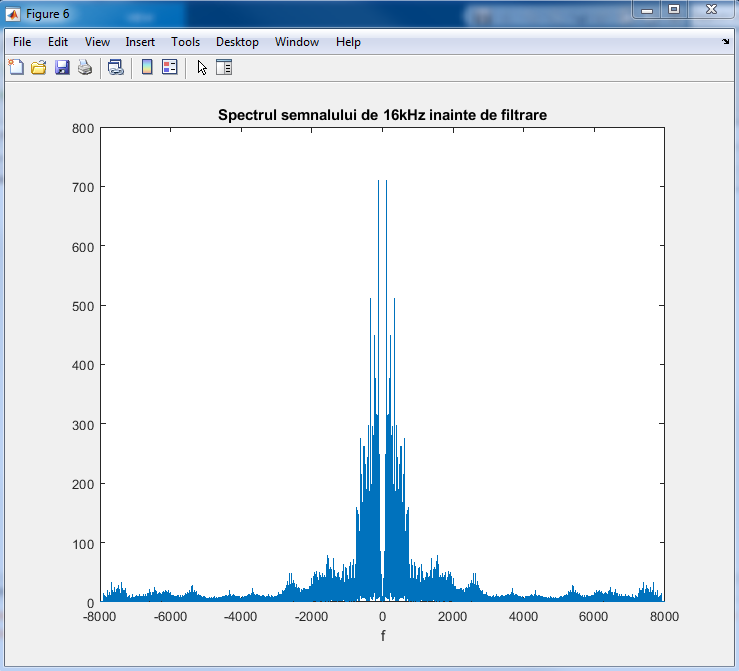
1. Apelarea funcției de downsampling.
2. Afișarea spectrului semnalului decimat:



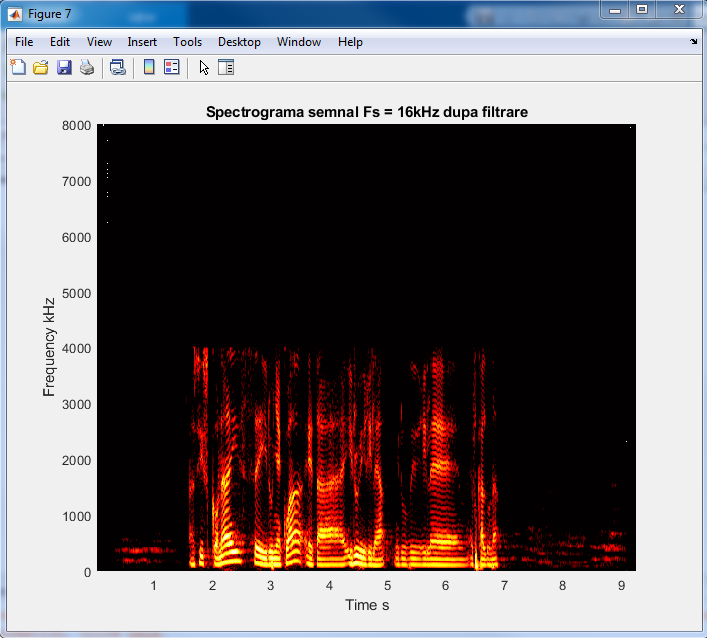
1. Afișarea spectrogramei semnalului decimat:



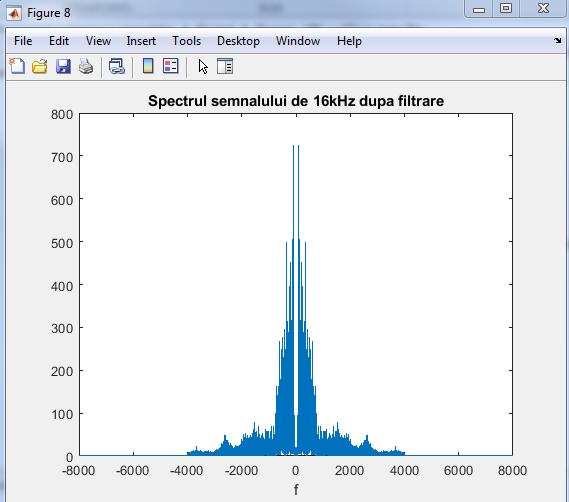
1. Apelarea funcției de upsampling
2. Afișarea spectrului semnalului upsampled înainte de filtrare:



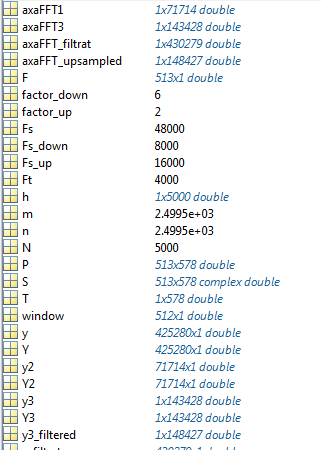
1. Filtrarea semnalului upsampled.
2. Afișarea spectrogramei semnalului upsampled după filtrare:



1. Afișarea spectrului semnalului upsampled după filtrare:



Pe tot parcursul procesării semnalului, am urmărit și datele din workspace, pentru a verifica implementarea corectă a codului și modificarea dimensiunilor vectorilor în funcție de operațiile pe care le-am aplicat.





## 5.3. Versiunea V02

În această versiune, codul pentru filtru este scris separat ca funcție, aceasta fiind doar apelată în fișierul principal. Codul pentru versiunea V02 se găsește în Anexa 2.

## 5.4. Versiunea V03 – explicații

Codul pentru versiunea V03 se găsește în Anexa 3.

Începând cu această versiune, am încercat să implementez metode de a verifica, cu ajutorul algoritmilor PEAQ și ViSQOL, evaluarea perceptuală a calității sunetului. Constrângerea care m-a determinat să caut aceste metode de implementare diferite a fost cerința impusă de ambii algoritmi, PEAQ și ViSQOL, ca frecvența de eșantionare a semnalului de referință să fie aceeași cu a semnalului alterat.

Așadar, în această versiune a proiectului am salvat, prima dată, semnalul original în vectorul y, apoi l-am mai salvat o dată în vectorul OrigTo16k. Primul, y, l-am procesat, cu downsample și upsample. Celui de-al doilea, OrigTo16k, i-am făcut direct downsampling la 16k, după ce l-am filtrat. Astfel, am comparat folosind algoritmii, semnalele OrigTo16k și y3\_filtered, ambele cu frecvența de eșantionare 16kHz.

Comanda pentru a asculta semnalul downsampled la 16kHz direct din semnalul original:

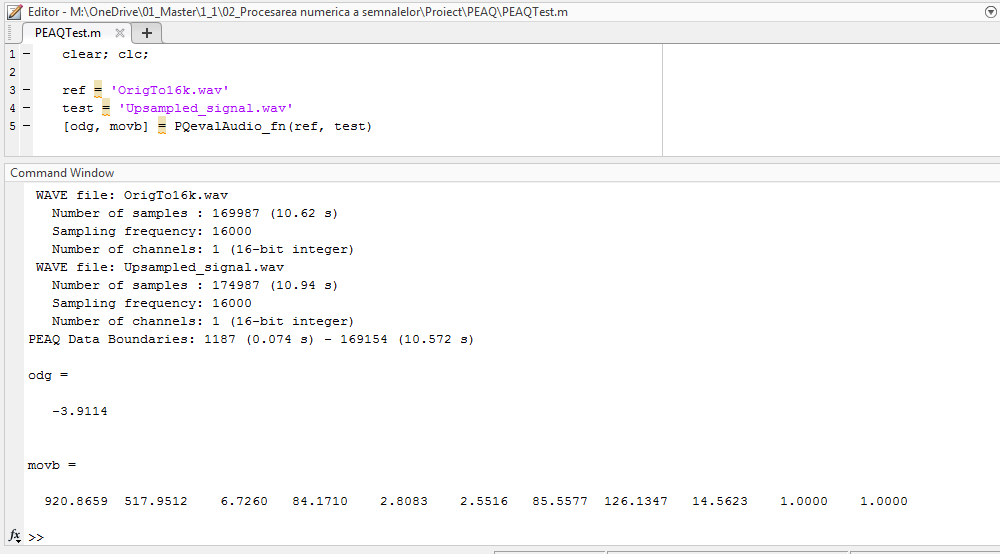
>> sound(OrigTo16kDownsampled, 16000)

Comanda pentru a asculta semnalul y3\_filtered, adică semnalul trecut prin downsampling la 8kHz apoi prin upsampling la 16kHz și filtrare:

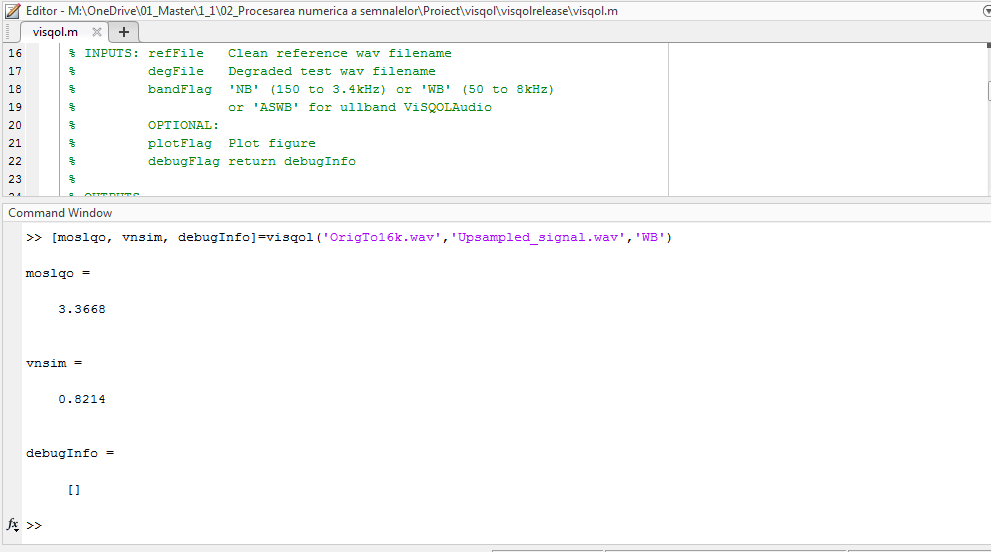
>> sound(y3\_filtered, Fs\_up)

Am comparat, prin urmare, folosind algoritmii, semnalele audio: OrigTo16k.wav și Upsampled\_signal.wav.

**Rezultatele** **PEAQ:**



**Rezultatele ViSQOL:**



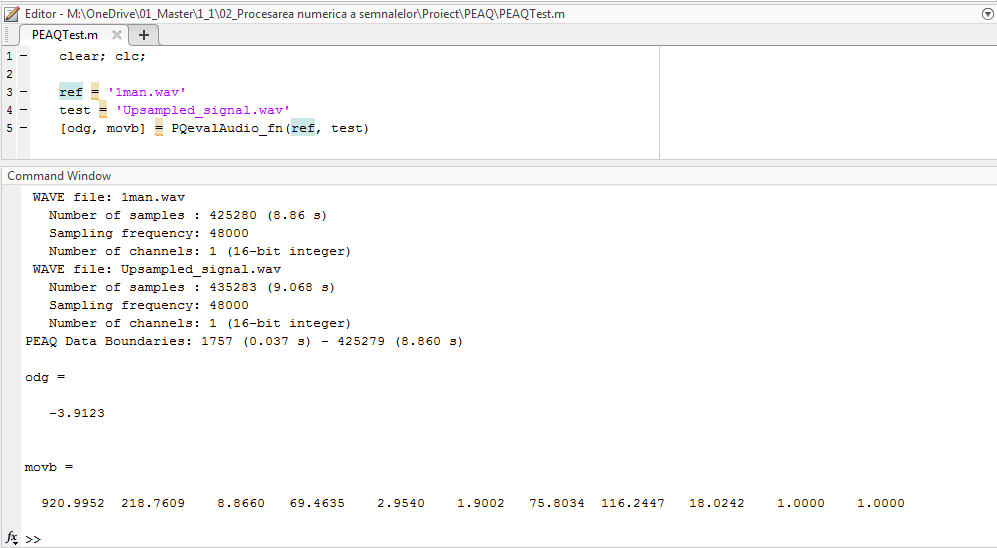
## 5.5. Versiunea V04 – explicații

Codul pentru versiunea V04 se găsește în Anexa 4.

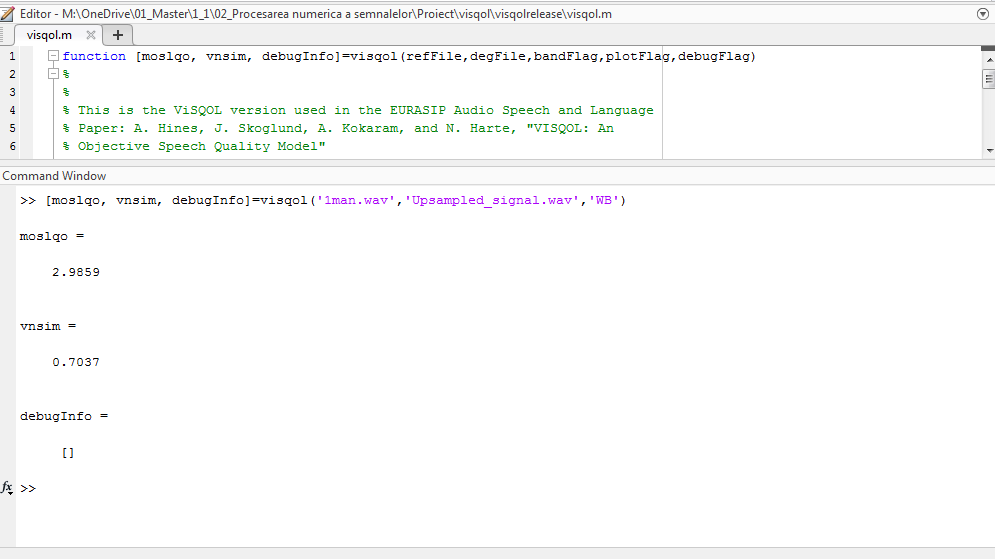
Explicația pentru care am creat această versiune este aceeași ca mai sus, nevoia de a avea aceeași frecvență de eșantionare la semnalele comparate.

În această versiune, am folosit un factor de upsampling de 6, astfel, după downsampling la 8kHz, semnalul ajunge direct la 48kHz prin upsampling și filtrare. Am făcut acest lucru pentru a compara semnalul final cu semnalul original, deși semnalul final are frecvența de eșantionare tot de 48kHz. Am comparat, prin urmare, folosind algoritmii, semnalele audio: 1man.wav și Upsampled\_signal.wav.

**Rezultatele PEAQ:**



**Rezultatele ViSQOL:**



## 5.6. Versiunea V05 – explicații + alegerea filtrului FTB

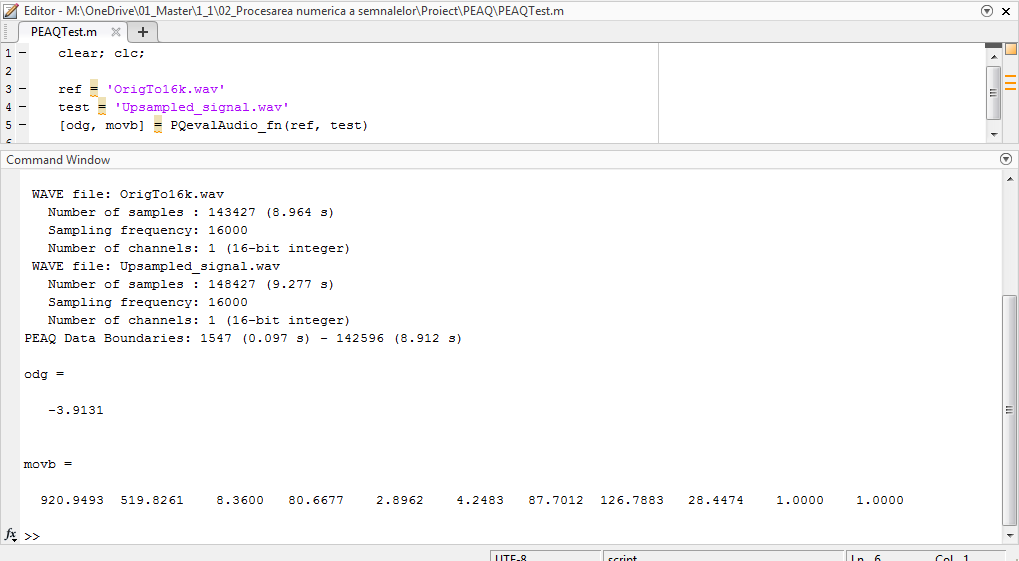
Codul pentru versiunea V05 se găsește în Anexa 5.

Această versiune are aceeași abordare ca versiunea V03. Am salvat, prima dată, semnalul original în vectorul y, apoi l-am mai salvat o dată în vectorul OrigTo16k. Primul, y, l-am procesat, cu downsample și upsample. Celui de-al doilea, OrigTo16k, i-am făcut direct downsampling la 16k, după ce l-am filtrat. Astfel, am comparat folosind algoritmii, semnalele OrigTo16k și y3\_filtered, ambele cu frecvența de eșantionare 16kHz.

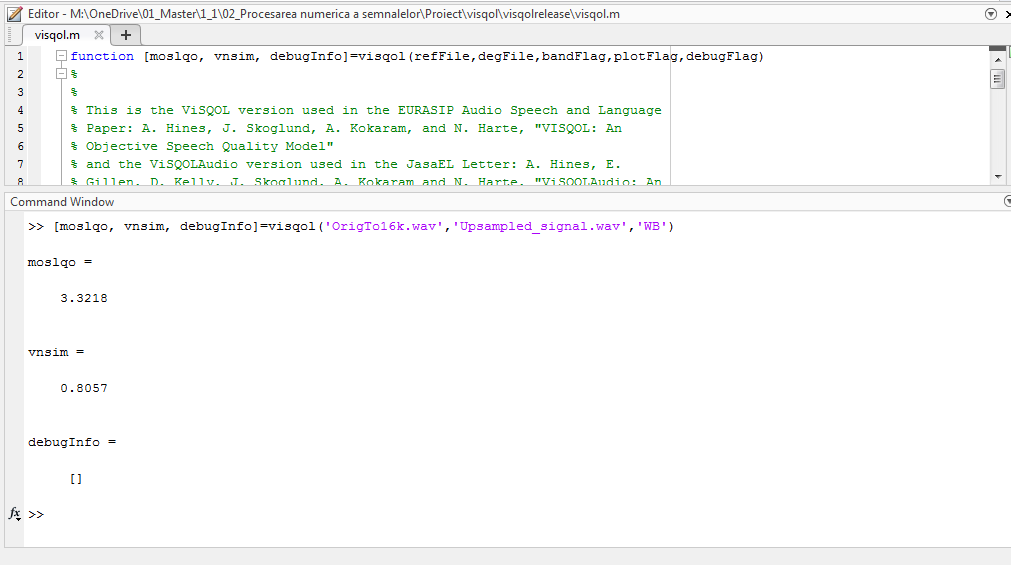
Diferența între cele două versiuni, V03 și V05, este aceea că la V05 am folosit un filtru trece-bandă, cu frecvențele de tăiere Ft1 = 30 Hz și Ft2 = 4000 Hz. Am realizat acest lucru pentru a vedea dacă intervine vreo diferență în rezultatele PEAQ și ViSQOL în urma eliminării componentelor de frecvență joasă.

Am comparat, prin urmare, folosind algoritmii, semnalele audio: OrigTo16k.wav și Upsampled\_signal.wav.

**Rezultatele PEAQ:**



**Rezultatele ViSQOL:**



# Concluzii

În concluzie, este evidențiată, prin utilizarea algoritmilor, informația care se pierde prin procesare, față de semnalul original. Deși vocea și mesajul transmis se aud clar în urma procesării, rezultatele algoritmilor sunt altele. Sunt rezultate medii, după părerea mea, indicând o degradare a semnalului, dar nu totală. De asemenea, utilizarea unui alt tip de filtru, și anume trece-bandă, față de cel trece-jos, nu aduce modificări majore în rezultatele evaluărilor perceptuale. De astfel, am ales aceste filtre în mod special deoarece sunt relativ ușor de implementat și mi s-au părut cele mai potrivite pentru aplicația proiectului.

De asemenea, am testat și alte semnale vocale, atât de bărbat, cât și de femeie, cu zgomot de fundal cât mai redus și voce clară, iar rezultatele evaluării perceptuale sunt aproape la fel cu cele prezentate în documentația de față.

# Anexe

**Anexa 1: Codul pentru versiunea V01 a proiectului**

**Proiect\_PNS.m**

clear all

close all

clc

[y,Fs] = audioread('1man.wav');

factor\_down = 6;

factor\_up = 2;

Fs\_down = Fs/factor\_down;

Fs\_up = Fs\_down \* factor\_up;

%afisare semnal original

figure(1)

axaFFT1=linspace(-Fs/2, Fs/2, length(y));

Y=fftshift(abs(fft(y)));

plot(axaFFT1,Y);

title('Spectrul semnalului original');

xlabel('f');

figure(2)

window=hamming(512); %%window with size of 512 points

[S,F,T,P] = spectrogram(y,window,256,1024,Fs,'yaxis');

surf(T,F,10\*log10(P),'edgecolor','none'); axis tight;view(0,90);

colormap("hot");

set(gca,'clim',[-80 -30]); %%clim is the limits of the axis colours

xlabel('Time s');

ylabel('Frequency kHz')

title ('Spectrograma semnalului original')

%FTJ

N = 5000;

Ft = 4000;

h = zeros(1,N);

for n=-(N-1)/2 : (N-1)/2

h(n+(N-1)/2 +1)=h(n+(N-1)/2+1)+(2\*Ft/Fs)\*sinc(2\*n\*Ft/Fs);

end

y\_filtrat = conv(y,h);

axaFFT\_filtrat=linspace(-Fs/2, Fs/2, length(y\_filtrat));

Y\_filtrat=fftshift(abs(fft(y\_filtrat)));

figure(3)

plot(axaFFT\_filtrat, Y\_filtrat);

title('Spectrul semnalului filtrat');

xlabel('f');

y2 = myDownsamplingFunction(y\_filtrat,Fs\_down,factor\_down);

audiowrite('Downsampled\_signal.wav',y2,Fs\_down);

figure(4)

axaFFT1=linspace(-Fs\_down/2, Fs\_down/2, length(y2));

Y2=fftshift(abs(fft(y2)));

plot(axaFFT1,Y2);

title('Spectrul semnalului decimat, Fs = 8kHz');

xlabel('f');

figure(5)

window=hamming(512); %%window with size of 512 points

[S,F,T,P] = spectrogram(y2,window,256,1024,Fs\_down,'yaxis');

surf(T,F,10\*log10(P),'edgecolor','none'); axis tight;view(0,90);

colormap("hot");

set(gca,'clim',[-80 -30]); %%clim is the limits of the axis colours

xlabel('Time s');

ylabel('Frequency kHz')

title ('Spectrograma semnal Fs = 8kHz')

y3 = myUpsamplingFunction(y2,Fs\_up,factor\_up);

figure(6)

axaFFT3=linspace(-Fs\_up/2, Fs\_up/2, length(y3));

Y3=fftshift(abs(fft(y3)));

plot(axaFFT3,Y3);

title('Spectrul semnalului de 16kHz inainte de filtrare');

xlabel('f');

%FTJ

N = 5000;

Ft = 4000;

h = zeros(1,N);

for m=-(N-1)/2 : (N-1)/2

h(m+(N-1)/2 +1)=h(m+(N-1)/2+1)+(2\*Ft/Fs\_up)\*sinc(2\*m\*Ft/Fs\_up);

end

y3\_filtered = conv(y3,h);

audiowrite('Upsampled\_signal.wav',y3\_filtered,Fs\_up);

figure(7)

window=hamming(512); %%window with size of 512 points

[S,F,T,P] = spectrogram(y3\_filtered,window,256,1024,Fs\_up,'yaxis');

surf(T,F,10\*log10(P),'edgecolor','none'); axis tight;view(0,90);

colormap(hot);

set(gca,'clim',[-80 -30]); %%clim is the limits of the axis colours

xlabel('Time s');

ylabel('Frequency kHz')

title ('Spectrograma semnal Fs = 16kHz dupa filtrare')

figure(8)

axaFFT\_upsampled=linspace(-Fs\_up/2, Fs\_up/2, length(y3\_filtered));

Y\_upsampled=fftshift(abs(fft(y3\_filtered)));

plot(axaFFT\_upsampled,Y\_upsampled);

title('Spectrul semnalului de 16kHz dupa filtrare');

xlabel('f');

**Anexa 2: Codul pentru versiunea V02 a proiectului**

clear all

close all

clc

[y,Fs] = audioread('1man.wav');

factor\_down = 6;

factor\_up = 2;

Fs\_down = Fs/factor\_down;

Fs\_up = Fs\_down \* factor\_up;

%afisare semnal original

figure(1)

axaFFT1=linspace(-Fs/2, Fs/2, length(y));

Y=fftshift(abs(fft(y)));

plot(axaFFT1,Y);

title('Spectrul semnalului original');

xlabel('f');

figure(2)

window=hamming(512); %%window with size of 512 points

[S,F,T,P] = spectrogram(y,window,256,1024,Fs,'yaxis');

surf(T,F,10\*log10(P),'edgecolor','none'); axis tight;view(0,90);

colormap("hot");

set(gca,'clim',[-80 -30]); %%clim is the limits of the axis colours

xlabel('Time s');

ylabel('Frequency kHz')

title ('Spectrograma semnalului original')

Ft = 4000;

y\_filtrat = LowPassFilter (Ft, y, Fs);

axaFFT\_filtrat=linspace(-Fs/2, Fs/2, length(y\_filtrat));

Y\_filtrat=fftshift(abs(fft(y\_filtrat)));

figure(3)

plot(axaFFT\_filtrat, Y\_filtrat);

title('Spectrul semnalului filtrat');

xlabel('f');

y2 = myDownsamplingFunction(y\_filtrat,Fs\_down,factor\_down);

audiowrite('Downsampled\_signal.wav',y2,Fs\_down);

figure(4)

axaFFT1=linspace(-Fs\_down/2, Fs\_down/2, length(y2));

Y2=fftshift(abs(fft(y2)));

plot(axaFFT1,Y2);

title('Spectrul semnalului decimat');

xlabel('f');

figure(5)

window=hamming(512); %%window with size of 512 points

[S,F,T,P] = spectrogram(y2,window,256,1024,Fs\_down,'yaxis');

surf(T,F,10\*log10(P),'edgecolor','none'); axis tight;view(0,90);

colormap("hot");

set(gca,'clim',[-80 -30]); %%clim is the limits of the axis colours

xlabel('Time s');

ylabel('Frequency kHz')

title ('Spectrograma semnal Fs = 8kHz')

y3 = myUpsamplingFunction(y2,Fs\_up,factor\_up);

figure(6)

axaFFT3=linspace(-Fs\_up/2, Fs\_up/2, length(y3));

Y3=fftshift(abs(fft(y3)));

plot(axaFFT3,Y3);

title('Spectrul semnalului de 16kHz inainte de filtrare');

xlabel('f');

Ft = 4000;

y3\_filtered = LowPassFilter(Ft,y3,Fs\_up);

audiowrite('Upsampled\_signal.wav',y3\_filtered,Fs\_up);

figure(7)

window=hamming(512); %%window with size of 512 points

[S,F,T,P] = spectrogram(y3\_filtered,window,256,1024,Fs\_up,'yaxis');

surf(T,F,10\*log10(P),'edgecolor','none'); axis tight;view(0,90);

colormap(hot); %%for the indexed colors, check this in help for black/white

set(gca,'clim',[-80 -30]); %%clim is the limits of the axis colours

xlabel('Time s');

ylabel('Frequency kHz')

title ('Spectrograma semnal Fs = 16kHz dupa filtrare')

figure(8)

axaFFT\_upsampled=linspace(-Fs\_up/2, Fs\_up/2, length(y3\_filtered));

Y\_upsampled=fftshift(abs(fft(y3\_filtered)));

plot(axaFFT\_upsampled,Y\_upsampled);

title('Spectrul semnalului de 16kHz dupa filtrare');

xlabel('f');

**Anexa 3: Codul pentru versiunea V03 a proiectului**

clear all

close all

clc

[y,Fs] = audioread('1man.wav');

[OrigTo16k, FOrigTo16k] = audioread('3man.wav');

OrigTo16k\_filtrat = LowPassFilter (8000, OrigTo16k, FOrigTo16k);

OrigTo16kDownsampled = myDownsamplingFunction(OrigTo16k\_filtrat,16000,3);

audiowrite('OrigTo16k.wav',OrigTo16kDownsampled,16000);

factor\_down = 6;

factor\_up = 2;

Fs\_down = Fs/factor\_down;

Fs\_up = Fs\_down \* factor\_up;

%afisare semnal original

figure(1)

axaFFT1=linspace(-Fs/2, Fs/2, length(y));

Y=fftshift(abs(fft(y)));

plot(axaFFT1,Y);

title('Spectrul semnalului original');

xlabel('f');

figure(2)

window=hamming(512); %%window with size of 512 points

[S,F,T,P] = spectrogram(y,window,256,1024,Fs,'yaxis');

surf(T,F,10\*log10(P),'edgecolor','none'); axis tight;view(0,90);

colormap("hot");

set(gca,'clim',[-80 -30]); %%clim is the limits of the axis colours

xlabel('Time s');

ylabel('Frequency kHz')

title ('Spectrograma semnalului original')

Ft = 4000;

y\_filtrat = LowPassFilter (Ft, y, Fs);

axaFFT\_filtrat=linspace(-Fs/2, Fs/2, length(y\_filtrat));

Y\_filtrat=fftshift(abs(fft(y\_filtrat)));

figure(3)

plot(axaFFT\_filtrat, Y\_filtrat);

title('Spectrul semnalului filtrat');

xlabel('f');

y2 = myDownsamplingFunction(y\_filtrat,Fs\_down,factor\_down);

audiowrite('Downsampled\_signal.wav',y2,Fs\_down);

figure(4)

axaFFT2=linspace(-Fs\_down/2, Fs\_down/2, length(y2));

Y2=fftshift(abs(fft(y2)));

plot(axaFFT2,Y2);

title('Spectrul semnalului decimat');

xlabel('f');

figure(5)

window=hamming(512); %%window with size of 512 points

[S,F,T,P] = spectrogram(y2,window,256,1024,Fs\_down,'yaxis');

surf(T,F,10\*log10(P),'edgecolor','none'); axis tight;view(0,90);

colormap("hot");

set(gca,'clim',[-80 -30]); %%clim is the limits of the axis colours

xlabel('Time s');

ylabel('Frequency kHz')

title ('Spectrograma semnal Fs = 8kHz')

y3 = myUpsamplingFunction(y2,Fs\_up,factor\_up);

figure(6)

axaFFT3=linspace(-Fs\_up/2, Fs\_up/2, length(y3));

Y3=fftshift(abs(fft(y3)));

plot(axaFFT3,Y3);

title('Spectrul semnalului de 16kHz inainte de filtrare');

xlabel('f');

Ft = 4000;

y3\_filtered = LowPassFilter(Ft,y3,Fs\_up);

audiowrite('Upsampled\_signal.wav',y3\_filtered,Fs\_up);

figure(7)

window=hamming(512); %%window with size of 512 points

[S,F,T,P] = spectrogram(y3\_filtered,window,256,1024,Fs\_up,'yaxis');

surf(T,F,10\*log10(P),'edgecolor','none'); axis tight;view(0,90);

colormap(hot);

set(gca,'clim',[-80 -30]); %%clim is the limits of the axis colours

xlabel('Time s');

ylabel('Frequency kHz')

title ('Spectrograma semnal Fs = 16kHz dupa filtrare')

figure(8)

axaFFT\_upsampled=linspace(-Fs\_up/2, Fs\_up/2, length(y3\_filtered));

Y\_upsampled=fftshift(abs(fft(y3\_filtered)));

plot(axaFFT\_upsampled,Y\_upsampled);

title('Spectrul semnalului de 16kHz dupa filtrare');

xlabel('f');

**Anexa 4: Codul pentru versiunea V04 a proiectului**

clear all

close all

clc

[y,Fs] = audioread('1man.wav');

factor\_down = 6;

factor\_up = 6;

Fs\_down = Fs/factor\_down;

Fs\_up = Fs\_down \* factor\_up;

%afisare semnal original

figure(1)

axaFFT1=linspace(-Fs/2, Fs/2, length(y));

Y=fftshift(abs(fft(y)));

plot(axaFFT1,Y);

title('Spectrul semnalului original');

xlabel('f');

figure(2)

window=hamming(512); %%window with size of 512 points

[S,F,T,P] = spectrogram(y,window,256,1024,Fs,'yaxis');

surf(T,F,10\*log10(P),'edgecolor','none'); axis tight;view(0,90);

colormap("hot");

set(gca,'clim',[-80 -30]); %%clim is the limits of the axis colours

xlabel('Time s');

ylabel('Frequency kHz')

title ('Spectrograma semnalului original')

Ft = 4000;

y\_filtrat = LowPassFilter (Ft, y, Fs);

axaFFT\_filtrat=linspace(-Fs/2, Fs/2, length(y\_filtrat));

Y\_filtrat=fftshift(abs(fft(y\_filtrat)));

figure(3)

plot(axaFFT\_filtrat, Y\_filtrat);

title('Spectrul semnalului filtrat');

xlabel('f');

y2 = myDownsamplingFunction(y\_filtrat,Fs\_down,factor\_down);

audiowrite('Downsampled\_signal.wav',y2,Fs\_down);

figure(4)

axaFFT1=linspace(-Fs\_down/2, Fs\_down/2, length(y2));

Y2=fftshift(abs(fft(y2)));

plot(axaFFT1,Y2);

title('Spectrul semnalului decimat');

xlabel('f');

figure(5)

window=hamming(512); %%window with size of 512 points

[S,F,T,P] = spectrogram(y2,window,256,1024,Fs\_down,'yaxis');

surf(T,F,10\*log10(P),'edgecolor','none'); axis tight;view(0,90);

colormap("hot");

set(gca,'clim',[-80 -30]); %%clim is the limits of the axis colours

xlabel('Time s');

ylabel('Frequency kHz')

title ('Spectrograma semnal Fs = 8kHz')

y3 = myUpsamplingFunction(y2,Fs\_up,factor\_up);

figure(6)

axaFFT3=linspace(-Fs\_up/2, Fs\_up/2, length(y3));

Y3=fftshift(abs(fft(y3)));

plot(axaFFT3,Y3);

title('Spectrul semnalului de 16kHz inainte de filtrare');

xlabel('f');

Ft = 4000;

y3\_filtered = LowPassFilter(Ft,y3,Fs\_up);

audiowrite('Upsampled\_signal.wav',y3\_filtered,Fs\_up);

figure(7)

window=hamming(512); %%window with size of 512 points

[S,F,T,P] = spectrogram(y3\_filtered,window,256,1024,Fs\_up,'yaxis');

surf(T,F,10\*log10(P),'edgecolor','none'); axis tight;view(0,90);

colormap(hot);

set(gca,'clim',[-80 -30]); %%clim is the limits of the axis colours

xlabel('Time s');

ylabel('Frequency kHz')

title ('Spectrograma semnal Fs = 48kHz dupa upsampling si filtrare')

figure(8)

axaFFT\_upsampled=linspace(-Fs\_up/2, Fs\_up/2, length(y3\_filtered));

Y\_upsampled=fftshift(abs(fft(y3\_filtered)));

plot(axaFFT\_upsampled,Y\_upsampled);

title('Spectrul semnalului de 16kHz dupa filtrare');

xlabel('f');

**Anexa 5: Codul pentru versiunea V05 a proiectului**

clear all

close all

clc

[y,Fs] = audioread('1man.wav');

[OrigTo16k, FOrigTo16k] = audioread('3man.wav');

OrigTo16k\_filtrat = BandPassFilter (30, 8000, OrigTo16k, FOrigTo16k);

OrigTo16kDownsampled = myDownsamplingFunction(OrigTo16k\_filtrat,16000,3);

audiowrite('OrigTo16k.wav',OrigTo16kDownsampled,16000);

factor\_down = 6;

factor\_up = 2;

Fs\_down = Fs/factor\_down;

Fs\_up = Fs\_down \* factor\_up;

%afisare semnal original

figure(1)

axaFFT1=linspace(-Fs/2, Fs/2, length(y));

Y=fftshift(abs(fft(y)));

plot(axaFFT1,Y);

title('Spectrul semnalului original');

xlabel('f');

figure(2)

window=hamming(512); %%window with size of 512 points

[S,F,T,P] = spectrogram(y,window,256,1024,Fs,'yaxis');

surf(T,F,10\*log10(P),'edgecolor','none'); axis tight;view(0,90);

colormap("hot");

set(gca,'clim',[-80 -30]); %%clim is the limits of the axis colours

xlabel('Time s');

ylabel('Frequency kHz')

title ('Spectrograma semnalului original')

Ft1 = 30;

Ft2 = 4000;

y\_filtrat = BandPassFilter (Ft1, Ft2, y, Fs);

axaFFT\_filtrat=linspace(-Fs/2, Fs/2, length(y\_filtrat));

Y\_filtrat=fftshift(abs(fft(y\_filtrat)));

figure(3)

plot(axaFFT\_filtrat, Y\_filtrat);

title('Spectrul semnalului filtrat');

xlabel('f');

y2 = myDownsamplingFunction(y\_filtrat,Fs\_down,factor\_down);

audiowrite('Downsampled\_signal.wav',y2,Fs\_down);

figure(4)

axaFFT1=linspace(-Fs\_down/2, Fs\_down/2, length(y2));

Y2=fftshift(abs(fft(y2)));

plot(axaFFT1,Y2);

title('Spectrul semnalului decimat');

xlabel('f');

figure(5)

window=hamming(512); %%window with size of 512 points

[S,F,T,P] = spectrogram(y2,window,256,1024,Fs\_down,'yaxis');

surf(T,F,10\*log10(P),'edgecolor','none'); axis tight;view(0,90);

colormap("hot");

set(gca,'clim',[-80 -30]); %%clim is the limits of the axis colours

xlabel('Time s');

ylabel('Frequency kHz')

title ('Spectrograma semnal Fs = 8kHz')

y3 = myUpsamplingFunction(y2,Fs\_up,factor\_up);

figure(6)

axaFFT3=linspace(-Fs\_up/2, Fs\_up/2, length(y3));

Y3=fftshift(abs(fft(y3)));

plot(axaFFT3,Y3);

title('Spectrul semnalului de 16kHz inainte de filtrare');

xlabel('f');

Ft1 = 30;

Ft2 = 4000;

y3\_filtered = BandPassFilter(Ft1,Ft2,y3,Fs\_up);

audiowrite('Upsampled\_signal.wav',y3\_filtered,Fs\_up);

figure(7)

window=hamming(512); %%window with size of 512 points

[S,F,T,P] = spectrogram(y3\_filtered,window,256,1024,Fs\_up,'yaxis');

surf(T,F,10\*log10(P),'edgecolor','none'); axis tight;view(0,90);

colormap(hot);

set(gca,'clim',[-80 -30]); %%clim is the limits of the axis colours

xlabel('Time s');

ylabel('Frequency kHz')

title ('Spectrograma semnal Fs = 16kHz dupa filtrare')

figure(8)

axaFFT\_upsampled=linspace(-Fs\_up/2, Fs\_up/2, length(y3\_filtered));

Y\_upsampled=fftshift(abs(fft(y3\_filtered)));

plot(axaFFT\_upsampled,Y\_upsampled);

title('Spectrul semnalului de 16kHz dupa filtrare');

xlabel('f');

# Bibliografie

* <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7056453/>
* <https://en.wikipedia.org/wiki/Aliasing>
* <https://www.hindawi.com/journals/ijmst/2009/758783/>
* <https://webspace.ulbsibiu.ro/catalina.neghina/Resurse/carti/indrumarPNS.pdf>
* <https://www.youtube.com/watch?v=--P0ZsbzJSw>
* <https://github.com/NikolajAndersson/PEAQ>
* <https://www.opticom.de/technology/peaq.php>
* <https://en.wikipedia.org/wiki/Perceptual_Evaluation_of_Audio_Quality>
* <https://en.wikipedia.org/wiki/Objective_difference_grade>
* <https://github.com/google/visqol/blob/master/README.md>
* <https://qxlab.ucd.ie/index.php/speech-quality-metrics/>
* <https://arxiv.org/pdf/2004.09584.pdf>
* <https://asmp-eurasipjournals.springeropen.com/articles/10.1186/s13636-015-0054-9>
* <https://github.com/stephencwelch/Perceptual-Coding-In-Python/blob/master/PEAQ.md>